COMPUTER SYSTEMS ORGANIZATION (CONT.)

THERE'S BEEN A LOT OF CONFUSION OVER 1024 vs 1000, KBYTE vs KBIT, AND THE CAPITALIZATION FOR EACH.

HERE, AT LAST, IS A SINGLE, DEFINITIVE STANDARD:

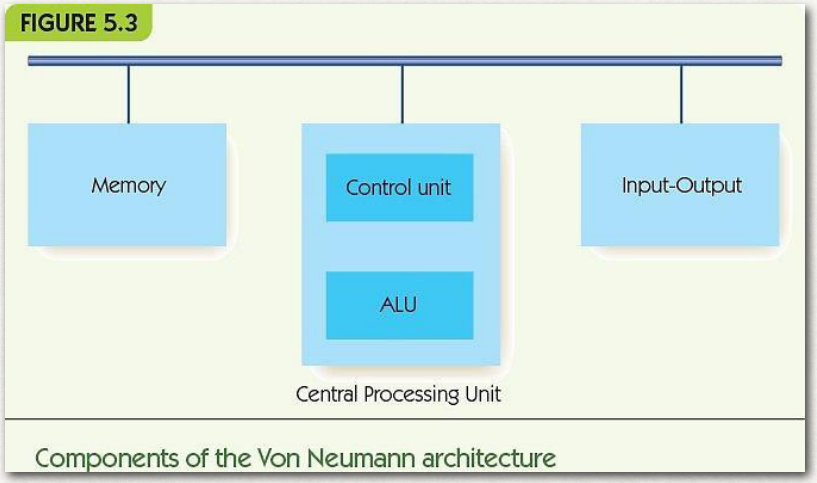
SYMBOL	NAME	SIZE	NOTES
kB	KILOBYTE	1024 BYTES OR 1000 BYTES	1000 BYTES DURING LEAP YEARS, 1024 OTHERWISE
KB	KELLY-BOOTLE STANDARD UNIT	1012 BYTES	COMPROMISE BETWEEN 1000 AND 1024 BYTES
KiB	IMAGINARY KILOBYTE	1024 √-1 BYTES	USED IN QUANTUM COMPUTING
kb	INTEL KILOBYTE	1023.937528 BYTES	CALCULATED ON PENTIUM FPU
Kb	DRIVEMAKER'S KILOBYTE	CURRENTLY 908 BYTES	SHRINKS BY 4 BYTES EACH YEAR FOR MARKETING REASONS
KBa	BAKER'S KILOBYTE	1152 BYTES	9 BITS TO THE BYTE SINCE YOU'RE SUCH A GOOD CUSTOMER

https://xkcd.com/394/

LEARNING OBJECTIVES

- Input and Output
- How the computer works - putting it all together
- Machine code
- The textbook's machine architecture
- Show the sequence of steps, using the book's notation, in the fetch, decode, and execute cycle to perform a typical instruction

VON NEUMANN ARCHITECTURE (FOR THE LAST TIME)



Components of the Von Neumann architecture

INPUT / OUTPUT

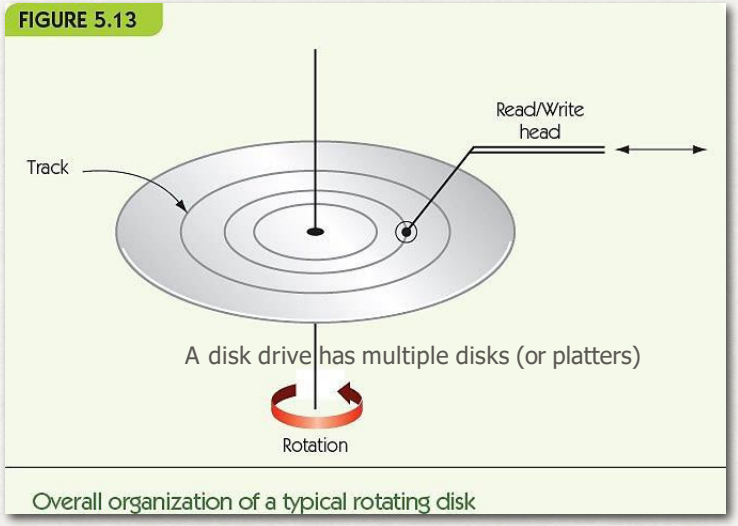
- **Input/output (I/O)** connects the processor to the outside world
  - Humans: keyboard, monitor, etc.
  - Data storage: hard drive, DVD, flash drive
  - Other computers: network
- RAM = **volatile memory** (gone without power)
- **Mass storage systems** = **nonvolatile memory**
  - **Direct access storage devices (DASDs)**
  - **Sequential access storage devices (SASDs)**

DIRECT ACCESS (RANDOM ACCESS)

DASDs

- Disks: Hard drives and optical media (CDs/DVDs)
  - **Tracks:** concentric rings around the disk surface
  - **Sectors:** fixed size segments of tracks, unit of retrieval
  - Time to retrieve data based on
    - **Seek time**
    - **Latency**
    - **Transfer time**
- Other nondisk DASDs: flash memory and solid-state drives (random access mass storage)

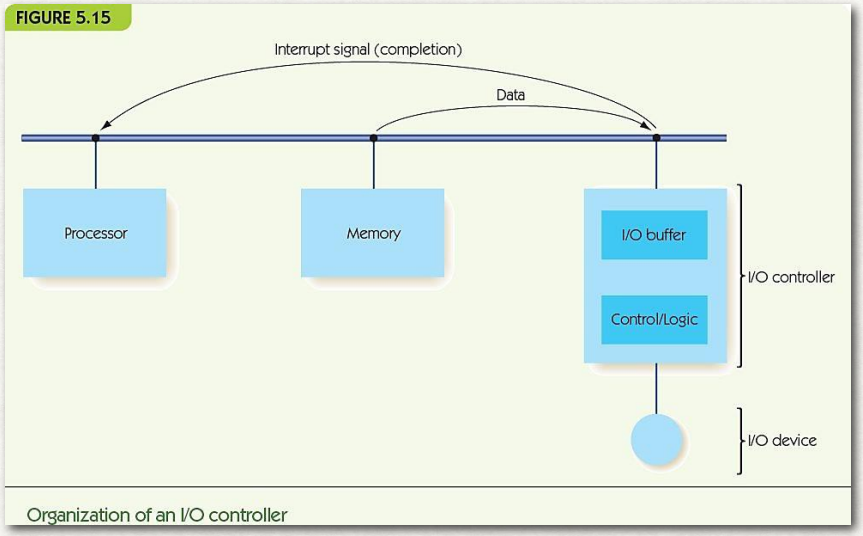
SPINNING DISK



WE THOUGHT MEMORY WAS SLOW

- DASDs and SASDs are orders of magnitude slower than RAM: (microseconds or milliseconds).
- **I/O Controller** manages data transfer with slow I/O devices, freeing processor to do other work.
  - DMA (Direct Memory Access) controllers mean that data can go to and from memory directly from a device without going through the processor (CPU)
- Controller sends an **interrupt signal** to processor when I/O task is done.

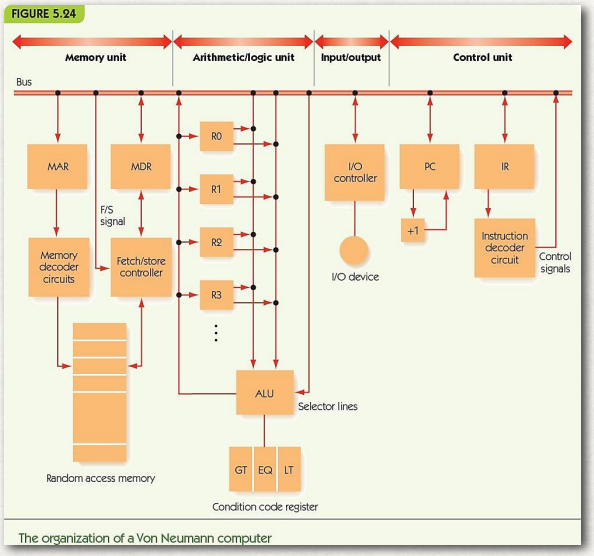
THE COMPONENTS OF A COMPUTER SYSTEM I/O AND MASS STORAGE



PUTTING THE PIECES TOGETHER

- Combine previous pieces: Von Neumann machine
- Fetch/decode/execute phase
  - Machine repeats until HALT instruction or error
  - Also called Von Neumann cycle
- Fetch phase: get next instruction from memory
- Decode phase: instruction decoder gets op code
- Execute phase: different for each instruction

OUR VON NEUMANN COMPUTER



The Control and ALU units are in the CPU

What are condition codes?

MACHINE CODE

- All instructions (opcodes) and operands (register, numbers or memory addresses) are stored as binary numbers
- If we write code like this we are working in **machine code**
- Machine code is immediately "understandable" to the computer but is a pain for humans
- e.g. If the opcode for ADD is 0001 and we have 16 registers we could encode the instruction ADD R5 to R6 and store the result in R2 as

opcode	operand 1	operand 2	operand 3
0001	0101	0110	0010

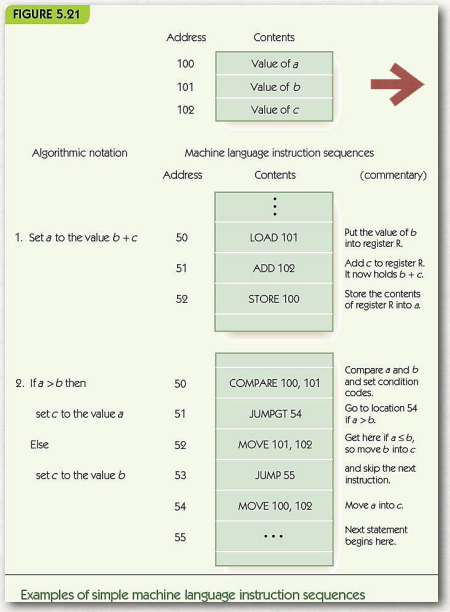
ASSEMBLY CODE

- If we have to work at the machine code level we can use mnemonics for opcodes and names for registers and memory locations
  - this is assembly code (more on this shortly), but the important thing is that it is mostly just a more human readable form of machine code
- An address is still just a number but we could use labels like X, Y or Z (or better still COUNT, TOTAL, ANSWER etc.) where each label corresponds to a number
- Because instructions and data are stored in the same memory in a Von Neumann architecture machine, we can label instruction addresses with labels too
- It is common to refer to registers as Rx, where x is a number from 0

TEXTBOOK MACHINE ARCHITECTURE

- There are only a small number (16) of instructions in the textbook machine language
- And only one register we can use as an operand
  - We can call this R, but in reality since there is only one of these registers we normally just assume we are using R
  - i.e. instead of saying ADD X,R meaning add the value of X to R we can say ADD X (and this will still mean add the value of X to R)

DATA AND INSTRUCTIONS



Examples of simple machine language instruction sequences

NOTATION FOR INSTRUCTIONS

Notation for computer's behavior

CON(A)	Contents of memory cell A
A → B	Send value in register A to register B (special registers: PC, MAR, MDR, IR, ALU, R, GT, EQ, LT, +1)
FETCH	Initiate a memory fetch operation
STORE	Initiate a memory store operation
ADD	Instruct the ALU to select the output of the adder circuit
SUBTRACT	Instruct the ALU to select the output of the subtract circuit

FETCH AND DECODE

Fetch phase

1. PC → MAR Send address in PC to MAR
2. FETCH Initiate fetch, data to MDR
3. MDR → IR Move instruction in MDR to IR
4. PC + 1 → PC Add one to PC

Decode phase

1. IR_{op} → instruction decoder (the IR_{op} means the opcode bits of the IR)

OUR INSTRUCTION SET

FIGURE 5.25

Binary Op Code	Operation	Meaning
0000	LOAD X	CON(X) → R
0001	STORE X	R → CON(X)
0010	CLEAR X	0 → CON(X)
0011	ADD X	R + CON(X) → R
0100	INCREMENT X	CON(X) + 1 → CON(X)
0101	SUBTRACT X	R – CON(X) → R
0110	DECREMENT X	CON(X) – 1 → CON(X)
0111	COMPARE X	if CON(X) > R then GT = 1 else 0 if CON(X) = R then EQ = 1 else 0 if CON(X) < R then LT = 1 else 0
1000	JUMP X	Get the next instruction from memory location X.
1001	JUMPGT X	Get the next instruction from memory location X if GT = 1.
1010	JUMPEQ X	Get the next instruction from memory location X if EQ = 1.
1011	JUMPLT X	Get the next instruction from memory location X if LT = 1.
1100	JUMPNEQ X	Get the next instruction from memory location X if EQ = 0.
1101	IN X	Input an integer value from the standard input device and store into memory cell X.
1110	OUT X	Output, in decimal notation, the value stored in memory cell X.
1111	HALT	Stop program execution.

Instruction set for our Von Neumann machine

EXECUTE EXAMPLES

Execution phase

LOAD X meaning CON(X) → R

1. IR_{addr} → MAR Send address X to MAR
2. FETCH Initiate fetch, data to MDR
3. MDR → R Copy data in MDR into R

What is the IR_{addr}?

What is R?

STORE X meaning R → CON(X)

1. IR_{addr} → MAR Send address X to MAR
2. R → MDR Send data in R to MDR
3. STORE Initiate store of MDR to X

MORE EXECUTE EXAMPLES

ADD X meaning R + CON(X) → R

1. IR_{addr} → MAR Send address X to MAR
2. FETCH Initiate fetch, data to MDR
3. MDR → ALU Send data in MDR to ALU
4. R → ALU Send data in R to ALU
5. ADD Select ADD circuit as result
6. ALU → R Copy selected result to R

Does this change the value at X?

JUMP X meaning get next instruction from X

1. IR_{addr} → PC Send address X to PC

What about PC + 1?

MORE EXECUTE EXAMPLES

COMPARE X meaning:

if CON(X) > R, then GT = 1, else 0

if CON(X) = R, then EQ = 1, else 0

if CON(X) < R, then LT = 1, else 0

this is more complicated than the compare for equality circuit we saw earlier

1. IR_{addr} → MAR Send address X to MAR
2. FETCH Initiate fetch, data to MDR
3. MDR → ALU Send data in MDR to ALU
4. R → ALU Send data in R to ALU
5. SUBTRACT Evaluate CON(X) – R

Sets EQ, GT, and LT

MORE EXECUTE EXAMPLES

JUMPGT X meaning:

if GT = 1, then jump to X,

else continue to next instruction

1. IF GT = 1 THEN $IR_{addr} \rightarrow PC$

Take some of the remaining instructions and see if you can produce the execute steps for them